POLIS (FRAMEWORK FOR HARDWARE-SOFTWARE CO-DESIGN OF EMBEDDED SYSTEMS)

POLIS (MARCO PARA HARDWARE-SOFTWARE CO-DISEÑO DE SISTEMAS EMBEBIDOS)

Luis Carlos Alvarez Alvarez
Asignatura de arquitectura de computadores
Escuela de Ingeniería de Sistemas e Informática
Universidad Industrial de Santander
Lcaa508@gmail.com

Andres Alfredo Bermudez Rodriguez
Asignatura de arquitectura de computadores
Escuela de Ingeniería de Sistemas e Informática
Universidad Industrial de Santander
Andruboy_3@hotmail.com

Abstract __ The codesign hardware / software is a modern technique for designing complex electronic systems consisting of hardware and software. This article presents an application of the co-design methodology in the implementation of fuzzy systems is shown. It is part of a formal description of fuzzy system implementation alternatives and different hardware / software are analyzed according to their cost and performance

Keywords --codesign, hardware, software.

Resumen — El codiseño hardware/software es una técnica moderna para el diseño de sistemas electrónicos complejos constituidos por hardware y software. En este artículo se muestra una aplicación de la metodología de co-diseño en la implementación de sistemas difusos. Se parte de una descripción formal del sistema difuso y se analizan diferentes alternativas de implementación hardware/software de acuerdo a su costo y desempeño

Palabras claves —codiseño;hardware;software.

I. INTRODUCCION

Desde los principios de la tecnología los sistemas electrónicos han venido experimentando un constante crecimiento en complejidad. Estos sistemas se han venido diseñando desde hace años con herramientas para realizar la compilación del software y la síntesis del hardware por lo que se tiene una necesidad de poder aplicar una forma más global de ingeniería automatizada al problema del codiseño .El término codiseño ha sido acuñado para describir el proceso de creación de sistemas mixtos que se componen de una porción de hardware y una porción de software y abarca las etapas de especificación, síntesis, estimación y verificación. El codiseño implica la especificación e implementación de un sistema a través del desarrollo simultáneo, concurrente e integrado de las partes software y hardware. POLIS es una herramienta de diseño enfocada a los sistemas embebidos que en conjunto con otros elementos de software hace los procesos de síntesis y verificación más manipulables

II. ESTADO DEL ARTE

Antes de comenzar con el desarrollo de las diferentes etapas de la metodología de Codiseño HW / SW es interesante analizar algunos conceptos relacionados con el diseño de sistemas digitales complejos como los sistemas embebidos Existen dos clases diferentes de sistemas: los de propósito general y los de propósito especial. Los primeros comprenden a las tradicionales computadoras que van desde las estaciones de trabajo a las computadoras, el POLIS sistema se centra alrededor de una representación única máquina similar de Estados Finitos. Un co-diseño de máquina de estados finitos (CFSM), como una clásica máquina de estados finitos, transforma un conjunto de elementos de entrada en un conjunto de salidas con sólo una cantidad finita de estado interno Los controladores embebidos para aplicaciones en tiempo real reactivos se implementan como sistemas de software-hardware mixto. Estos controladores utilizan microprocesadores, microcontroladores y procesadores de señales digitales, pero no se utilizan ni perciben como ordenadores.
En general, el software se utiliza para características y flexibilidad, mientras que el hardware se utiliza para el rendimiento. Algunos ejemplos de aplicaciones de controladores embebidos son:

- *Electrónica de consumo:* los hornos de microondas, cámaras, reproductores de discos compactos.
- *Telecomunicaciones:* centrales telefónicas, teléfonos celulares.
- *Automoción* controladores de motores, controladores de frenos antibloqueo.
- *Control de la Planta:* robots, monitores de plantas.

III. SISTEMAS EMBEBIDOS

como se dijo anteriormente existen dos tipos de sistema entre ellos los de propósito general y los de propósito especial Los de propósito general comprenden a las tradicionales computadoras que van desde las estaciones de trabajo a las computadoras personales de mano. Todas estas, pueden ser programadas por un usuario para ser utilizadas en diferentes
aplicaciones y han sido diseñadas para aplicaciones generales, lo cual generó que determinadas tareas que deben ser ejecutadas con restricciones (como, por ejemplo tiempo de ejecución) no puedan realizarse eficientemente. Surgen desde ese entonces los sistemas dedicados (los de propósito especial). Estos, se diseñan para cumplir una tarea específica fija. La mayoría es configurada una sola vez y luego funciona independientemente. El usuario tiene un acceso limitado a programar estos últimos sistemas.

- Entre los sistemas de propósito especial existen los sistemas embebidos, los cuales pueden ser caracterizados con los siguientes rasgos:

  - son usualmente embebidos en otros productos, funcionan por cuenta propia,
  - son infrecuentemente reprogramados. Su funcionalidad es mayormente fija,
  - funcionan frecuentemente de modo reactivo, respondiendo a estímulos (entradas) externos,
  - son implementados por numerosos procesos concurrentes requiriendo una comunicación entre procesadores,
  - tienen severos requerimientos de tiempo, por ej. restricciones de tiempo real,
  - son utilizados en forma intensiva respecto a las entradas – salidas (E-S),
  - son extremadamente sensibles respecto a criterios de costo, potencia, y performance,
  - tiene fuertes restricciones de fiabilidad y exactitud.

Los controladores embebidos están dedicados a funciones de control y reaccionan a eventos externos. Responden a estímulos del exterior cambiando su estado interno y produciendo resultados de salida. Normalmente, soportan un conjunto de modos y ajustes y sus restricciones de tiempo real suelen ser del orden de los milisegundos. Por lo tanto, su desempeño requerido varía desde bajo a moderado. Por esta razón, microcontroladores o incluso procesadores de propósito general (8086, 6800, etc.) son frecuentemente suficiente para implementar un controlador embebido. Los sistemas embebidos dominados por flujo de datos se dedican a comunicaciones y procesamiento de datos. Por lo tanto, son frecuentemente llamados sistemas transformacionales. Estos sistemas suelen ejecutar (a tiempo real) una función especial en una ventana de tiempo predefinido. Requieren un desempeño mucho mayor que los controladores embebidos. Por lo tanto microcontroladores no son suficiente y microprocesadores mucho más potentes (muy frecuentemente DSPs o ASIPs) y ASICs son requeridos.

Los métodos actuales para el diseño de sistemas embebidos requieren que se especifique y diseño de hardware y software por separado. Una especificación, a menudo incompleta y escrita en lenguas no oficiales, se desarrolla y se envía a los ingenieros de hardware y software. Partición de hardware-software se decidió *a priori* y se adhiere a tanto como es posible, ya que cualquier cambio en esta partición pueden hacer necesario un extenso rediseño. Los diseñadores a menudo se esfuerzan por hacer que todo encaje en el software, y sin carga sólo algunas partes del diseño de hardware para cumplir con las restricciones de tiempo. Los problemas con estos métodos de diseño son:

- La falta de una representación hardware-software unificada, lo que conduce a dificultades en la verificación de todo el sistema, y por lo tanto a incompatibilidades través del límite de HW / SW.
- *A priori* definición de particiones, que conduce a diseños sub-óptimos.
- tiempo de salida al mercado de la falta de un flujo de diseño bien definido, lo que hace difícil Revisión de la especificación, y afecta directamente.

IV. CODISEÑO Y FASE DE SU METODOLOGIA

Gran parte de los sistemas electrónicos digitales actuales están constituidos por componentes de hardware y de software. Los componentes de software son programas desarrollados en lenguajes de alto nivel que corren sobre un procesador programable.
Los modelos y herramientas de co-diseño se diferencian por el nivel donde se puede dar la reprogramación. Cuando ésta se da en el nivel de aplicación de un procesador, el co-diseño consiste en el tratamiento unificado y la síntesis automática del software y del hardware, asumiendo este último como un co-procesador

Una definición formal de codiseño HW / SW se da en el proceso de diseño de un sistema que combina las perspectivas hardware y software desde los estados primarios, para aprovechar la flexibilidad del diseño y la localización eficiente de las funciones. En el marco del codiseño HW / SW, la concepción de los sistemas hardware y del sistema software es una sola, completamente diferente a la del proceso del diseño tradicional, donde generalmente se desarrolla el hardware y posteriormente se adapta el software a ese diseño, lo cual en muchas oportunidades resulta ineficiente.

El ciclo de diseño es mostrado en la Figura 1. Los pasos principales de la metodología se describen a continuación:

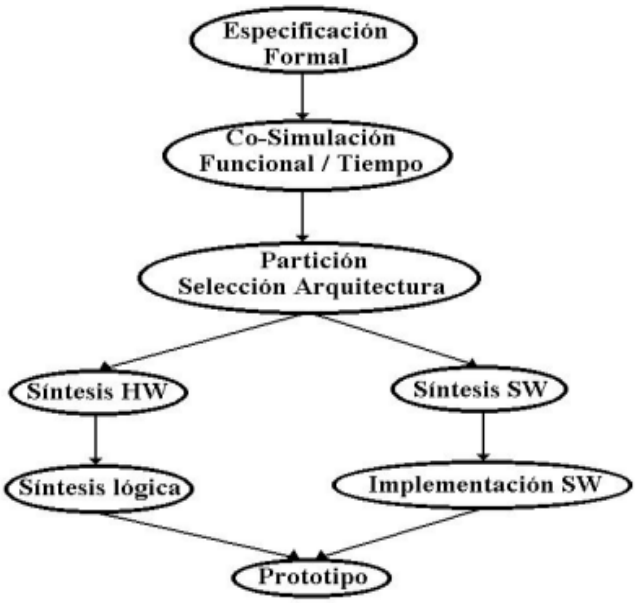


Figura 1. Flujo de co-diseño con Polis

- *Especificación formal*

Se parte de una especificación del sistema a implementar en un lenguaje de alto nivel. El lenguaje de especificación recomendado para Polis es **Esterel**, que es un lenguaje de programación sincrónica de sistemas reactivos con un modelo de concurrencia con sincronía perfecta, en el cual procesos concurrentes pueden ejecutar tareas e intercambiar información en tiempo cero, por lo menos a nivel conceptual. Este lenguaje permite especificar la funcionalidad independientemente de la implementación hardware/software.

- *Co-simulación*

Polis ofrece un entorno para evaluar los compromisos y alternativas de diseño mediante simulación. Los modelos, ya sean implementados en hardware o software son simulados empleando Ptolemy. El cual es un ambiente de simulación y diseño de sistemas heterogéneos. En Ptolemy, los objetos son descritos en diferentes niveles de abstracción (dominios). Las primitivas de cada dominio, llamadas "estrellas", pueden ser usadas en modo de simulación o en modo síntesis. La función de cada una de ellas se describe en **Esterel**. Para sistemas reactivos, el dominio más adecuado es el dominio DE (Discret Event), que usa un modelo de computación activado por eventos . La co-simulación puede hacerse a dos niveles de abstracción: - nivel funcional, donde el tiempo es ignorado y sólo importa la correcta operación del sistema; y - nivel de tiempo aproximado, en la cual el tiempo de ejecución se estima mediante el cálculo de ciclos de ejecución para el software y el hardware según un reloj maestro.

- *Partición del diseño*

En el proceso de co-simulación se puede elegir dinámicamente el tipo de implementación de cada componente y otros parámetros como el procesador, la velocidad de reloj, el algoritmo de scheduling, con el fin de satisfacer las restricciones impuestas sobre el sistema. Un aspecto clave de la metodología de Polis es que permite explorar interactivamente una gran variedad de alternativas de partición, estimando el costo y el desempeño de cada una de ellas.

- *Co-síntesis*

La co-síntesis se refiere a la síntesis integrada de las particiones de hardware y software. Cada módulo descrito en Esterel tiene asociada una máquina de estados finitos de co-diseno (CFSM) Las CFSMs son localmente síncronas y la interacción entre ellas es globalmente asíncrona. Cada CFSM se puede sintetizar en hardware o software. Para la síntesis de software Polis traslada cada CFSM a una representación intermedia de alto nivel independiente de la tecnología, conocida como S-GRAPH; luego el SGRAPH se traduce a código C estándar que puede ser llevado directamente a un procesador mediante compilación.

Polis sintetiza un pequeño sistema operativo que se encarga de asignar puertos de entrada y salida para la comunicación del software con el hardware, realizar el scheduling de las tareas de software de acuerdo con un algoritmo que se selecciona durante la partición. La síntesis incluye la generación de código en lenguaje C con el scheduler y los drivers de I/O.

Dentro del proceso de síntesis POLIS genera, para el hardware, una descripción lógica de circuitos digitales sincrónicos con formato BLIF, la cual puede ser usada para programar FPGAs o ser convertida a código VHDL. Con los resultados de la síntesis se puede construir un prototipo del sistema diseñado mediante la conexión de un procesador y una FPGA programados

Al momento de llevar a la practica las técnicas de codiseño aplicadas al desarrollo de sistemas embebidos , se hace necesario contar con herramientas unificadas que soporten todas las actividades que involucra este proceso. Son varios los grupos de investigación abocados al área del codiseño HW / SW como en este caso es fundamental la herramienta POLIS en la arquitectura Multi-procesador,

El diseño se realiza en un marco unificado, POLIS , con una representación unificada de hardware y software, con el fin de perjudicar ni la implementación de hardware ni de software. Este modelo se mantiene durante todo el proceso de diseño, con el fin de preservar las propiedades formales de diseño.

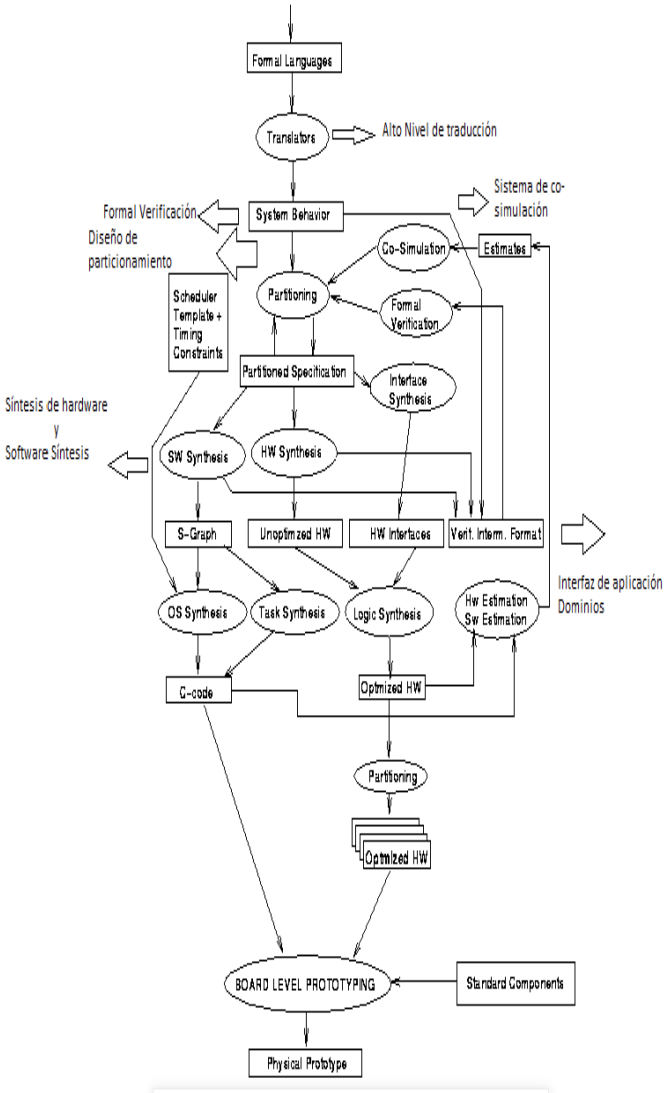
Aproximaciones al codiseño			
Modelado de sistemas			Ptolemy
			Simulink
Implementación heterogénea	Síntesis de procesadores		Castle
	Arquitectura mono-procesador	ASIC simple	Cosyma
			Lycos
		ASICs múltiples	Mickey
			Tosca
			Vulcan
	Arquitectura multi-procesador		Chinook
			Cool
			Cosmos
			CoWare
			Polis
			SpecSyn

Las distintas aproximaciones al codiseño pueden ser comparadas en función de su poder de especificación y el poder de implementación. En particular el poder de especificación clasifica a las diferentes herramientas concerniente a:
- estilo de modelado: homogéneo o heterogéneo,
- posibilidad de validación: simulación o verificación,
- dominio de aplicación: flujo de control, flujo de datos o ambos.

Una implementación de hardware sincrónica de CFSM puede ejecutar una transición en 1 ciclo de reloj, mientras que una implementación de software requerirá más de 1 ciclo de reloj.

CFSMS son también un modelo sintetizable y verificable, porque muchas teorías existentes y herramientas para el modelo FSM se pueden adaptar fácilmente para CFSM.

El caudal de diseño que se implementa actualmente en el **POLIS** sistema se representa en la siguiente figura y se describe con más detalle a continuación "Ref. [1]"



1. *Alto Nivel de traducción*
   En **POLIS** , diseñadores escribir sus especificaciones en un lenguaje de alto nivel (por ejemplo, ESTEREL, FSM gráficas, subconjuntos de Verilog o VHDL) que puede traducirse directamente en CFSMS. Cualquier lenguaje de alto nivel con la semántica precisa sobre la base de las MEF prolongados se puede utilizar para modelarCFSMS individuales (Actualmente estérel se apoya directamente).

2. *Formal Verificación*
   La metodología de especificación formal y síntesis incrustado dentro de **POLIS** hace posible interactuar directamente con los algoritmos de verificación formal que se basan en FSMs existente. **POLIS** incluye un traductor de la CFSM al formalismo FSM que puede ser alimentado directamente a los sistemas de verificación (por ejemplo, **VIS**). Además de descubrir errores en un diseño, también utilizamos la verificación formal de guiar el proceso de síntesis. Las herramientas de verificación formal hoy

en día todavía tienen problemas con la complejidad. Hemos desarrollado una metodología que incorpora un conjunto de reglas de extracción y la asunción específicas para **POLIS** y CFSMS. Con esta metodología de verificación formal somos capaces de verificar los diseños que son más grandes de lo que es posible anteriormente.

3. *Sistema de co-simulación*
   La co-simulacion de un nivel de sistema HW-SW es una manera de dar a los diseñadores retroalimentación sobre sus opciones de diseño. Estas opciones de diseño incluyen HW-SW partición, la selección de la CPU, y la selección del planificador. rápida cronometrada co-simulación (hasta millones de ciclos de reloj por segundo en una estación de trabajo) es posible en **POLIS** gracias a las técnicas de síntesis de software y de estimación de rendimiento que se describen a continuación. Actualmente **PTOLOMEO** es como un motor de simulación. Código VHDL incluyendo toda la información co-simulación es también una salida del sistema, por lo que cualquier simulador VHDL comercial se puede adaptar para este fin.

4. *Diseño de particionamiento*
   mediante la partición de diseño que significa tomar decisiones de diseño a nivel de sistema, tales como HW-SW partición, la selección arquitectura objetivo, y la selección del planificador. Estas decisiones se basan en gran medida en la experiencia de diseño y son muy difíciles de automatizar. Por lo tanto presentamos el diseño con un ambiente para evaluar rápidamente cualquier decisión a través de diversos mecanismos de retroalimentación, ya sea la verificación formal o sistema de co-simulación.

5. *Síntesis de hardware*
   Una CFSM sub-red elegida para la aplicación HW se implementa y optimiza el uso de técnicas de síntesis lógica del **SIS** . Cada CFSM, interpretado como una especificación de nivel de trasferencia entre registros, se puede mapear en **BLIF, XNF, VHDL o Verilog**.

6. *Software Síntesis*
   Un CFSM sub-red elegida para la aplicación SW está asignada a una determinada estructura de software que incluye un procedimiento para cada CFSM, junto con un sencillo en tiempo real del sistema operativo:

   ❖ CFSMS. El comportamiento reactivo se sintetiza en un proceso de dos pasos:

   - Implementar y optimizar el comportamiento deseado en un alto nivel de representación, independiente del procesador del proceso de toma similar a un diagrama de flujo de control / datos.

   - Traducir el gráfico de flujo de control / datos en código C portátil y utilizar cualquier compilador disponibles para implementar y optimizar en un conjunto de instrucciones específicas, micro-controlador-dependiente.

"Un estimador de tiempo analiza rápidamente las características del programa y los informes de tamaño de código y velocidad. El algoritmo utiliza una fórmula, con parámetros obtenidos a partir de los programas de referencia, para calcular el retardo de cada nodo en el gráfico de flujo de control / de datos para varias arquitecturas de micro-controlador (datos de caracterización para MIPS R3000 y Motorola 68HC11 y 68.332 ya están disponibles).
El método se utiliza para sincronizar los bloques de hardware y software dentro del entorno de co-simulación basada en PTOLOMEO."

   ❖ Sistema operativo en tiempo real. Un sistema operativo específico de la aplicación, que consiste en un planificador (por ejemplo Rate-monótona y Límite-monótona) y de E / S de los conductores, se genera para cada diseño de particiones.

7. *Interfaz de aplicación Dominios*
   interfaces entre diferentes dominios de aplicación (hardware y software) son sintetizados de forma automática dentro **POLIS**. Estas interfaces se presentan en forma de circuitos y procedimientos de software (controladores de E / S) incrustados en la aplicación sintetizado cooperar. La comunicación puede ser a través de los puertos de E / S disponibles en el micro controlador, o la memoria general de E / S mapeada.

V. SOBRE POLIS Y LA SEMANTICA DE TEMPORIZACION

La conectividad del sistema y la comunicación de un sistema es típicamente modelada como una red de módulos que se comunican. Hay que definir la semántica intra e inter-módulo se pueden resumir en dos componentes: el de datos, que se almacena en una memoria intermedia común y el momento, que da información sobre cuándo se lee y se escribe estos datos.

Sincronía

- *El Modelo*

  La hipótesis síncrona supone que *los cálculos toman el tiempo cero.* Es decir, en un punto en el tiempo llamado un instante, un módulo síncrono lee sus entradas, lleva a cabo su cálculo, y escribe sus salidas a la vez. El *tiempo entre los instantes es infinita*, ya que no hay interacción entre los instantes (entrada de *n* llega un tiempo infinitamente largo tiempo después de la entrada de *n-1*). El comportamiento de un sistema síncrono es una corriente de pares de entrada / salida, donde se asocia un par con cada instante.

  La comunicación en un sistema síncrono implica entonces que todos los módulos de lectura y escritura a una memoria común compartida. El tiempo de lectura / escritura es cero, mientras que el tiempo entre la lectura / escritura es infinita.

- *Implicaciones del Modelo*

  En una implementación real, los cálculos no toman realmente el tiempo cero, ni es el retraso entre los cálculos infinitos. Un cálculo toma el tiempo despreciable con respecto a las acciones que tienen lugar en su entorno. Por lo tanto, el medio ambiente ve un módulo que calcula en el tiempo cero, y el módulo ve un ambiente que requiere tiempo infinito en entre valores de entrada. Esta suposición sobre los retrasos relativos entre el módulo y el medio ambiente debe ser confirmada en la puesta en práctica sintetizada. En cuanto al hardware, el reloj síncrono mundial se debe ejecutar lo suficientemente lento para cada módulo para completar su cálculo en relación con el reloj que avanza. En el software, un módulo síncrono que es llamado por el sistema operativo debe ser permitido para completar su reacción antes de que sus entradas se cambian (por ejemplo, por otro módulo).

- *Ordenamiento*

  No hay orden de lectura de las entradas, cálculos, o la escritura de las salidas (excepto que una salida sólo se escribe después de los cálculos de los que depende se han realizado, y un cálculo sólo se realiza después de que las entradas sobre de los que depende hayan sido leídos). Un ordenamiento puede ser impuesta, pero esto no es parte del modelo.

Esterel

Como lo habíamos mencionado anteriormente nos apoyamos mucho en Esterel donde es lenguaje sincrónico y un compilador para ese idioma. Un sistema de Esterel se compone de interactuar módulos síncronas. La Comunicación entre módulos es a través de *señales*. Cada módulo tiene un conjunto de señales de entrada y de salida, que se leen y se escriben de forma instantánea y sin ningún orden en particular (es decir, los emisores y receptores de esas señales pueden no depender de un orden en particular)

Asincrónico

- *El Modelo*

  En un modelo asíncrono, hay suposiciones hechas sobre el timing(ritmo), relativo o absoluto, de cualquier señal, las variables o cálculos son más complejos. En la forma más pura de asincronía, llamado *insensibles al retardo,* el objetivo es generar un diseño correcto independientemente de los retrasos en los elementos de cálculo (en el hardware, puertas) o canales de comunicación (en hardware, cables). Se ha demostrado que no hay circuitos de interés práctico se pueden diseñar sin restricciones más severas a la modalidad o una proliferación de elementos computacionales. En hardware, esta última forma de retraso puede ser puro o inercial, limitado o ilimitado, número entero o continua, aplicada a los cables de retroalimentación o alambres no de retroalimentación o puertas. El caso más restrictivo es precisamente el de sincronía: los retrasos son fijos y conocidos para cada puerta y el cable, y hay un elemento de retardo infinito en (al menos) cada cable de realimentación.

- *Implicaciones del modelo*

  Un modelo típico de retardo en un sistema asíncrono asigna a cada puerta y / o alambre de un retardo inercial-up delimitada: esto permite un retardo variable, y permite el modelado de pulsos de eventos perdidos, ambos de los cuales son necesarios para modelar sistemas reales.

  La noción de retraso es completamente suelto. La comunicación es basada en eventos ya que no hay reloj global con el que desea sincronizar diferentes módulos. Por lo tanto, no pasa nada en un sistema asíncrono o módulo hasta que algún acontecimiento desencadena una reacción. Esta reacción tiene una cantidad no especificada de

tiempo, y da salida a los eventos que desencadenan reacciones adicionales.

- ✓ ¿Por qué utilizar Sincronía? Y ¿Asincrónica?

  Haciendo una comparación entre que modelo utilizar para el sistema tenemos que tener en cuenta varios aspectos. El modelo síncrono se refiere a dos tipos de retardo: *cero,* donde todo ocurre a la vez, e *infinitas*, donde las acciones están completamente separados.

  Los asíncronos modelo trata de un tipo de retardo que es a la vez *distinto de cero* y *finita.* Es típicamente hasta acotada. Las diferencias en las simulaciones de dos sistemas, donde uno están asumiendo la comunicación síncrona y asíncrona del otro, se pueden atribuir a estas tres diferentes nociones de demora.

  Un sistema síncrono puede ser más lenta, ya que un "reloj" se debe ejecutar a la velocidad del módulo más lento.

- ✓ ¿Por qué utilizar la sincronía?

  Es más fácil de diseñar y verificar módulos, muchos sistemas no son de tiempo crítico y puede permitirse una aplicación síncrona, y muchos sistemas son de hecho naturalmente síncrona.

- ✓ ¿Por qué utilizar la asincronía?

  El mundo real es asíncrona y por lo tanto cualquier interfaz que será asíncrona, para aplicaciones en tiempo o energía crítica de un solo reloj mundial es inaceptable, y para algunos sistemas, como los sistemas heterogéneos distribuidos, las restricciones a la sincronía son demasiado complicados (por ejemplo, la misma estructura de control debe ejecutarse en todos los procesadores, y los cálculos sólo de datos totalmente independientes de control puede distribuirse).

POLIS

- ➢ *El Modelo*

  POLIS utiliza el modelo de los galones de temporización y sincronización. Es decir, una red POLIS consta de nodos (llamados CFSMS) de módulos síncronos que interactúan a través de una red asíncrona. Dicho de otra manera a nivel global la parte asincronica es la red, y la parte local sincrónico es el conjunto de nodos síncronos.

- ➢ *Implicaciones del Modelo*

  Sincronía local significa que cada módulo lee / computa/escribe como si estuviera en una red síncrona: en el tiempo cero, y con un conjunto de entrada /salida no ordenada. asincronía Global significa que tan pronto como un módulo se detiene para esperar una señal, el retraso en espera es completamente indeterminado. En el mundo totalmente sincrónico, un retardo atómico es un instante, y todos los módulos se retrasan en la misma cantidad. Con asincronía global, un módulo de espera hasta que es de nuevo disparado por una de sus entradas. Esta activación puede ocurrir en cualquier momento en el futuro.

  El funcionamiento del sistema GALS procede como sigue. La red asíncrona determina cuándo llamar a cada nodo para hacer un cálculo. Esto es por eventos. Una vez que un nodo se llama, su reacción es sincrónico:

  Se lee, calcula, y escribe a la vez, devolver el control a la red asíncrona. Visto desde la red, esto lleva tiempo: entradas se leen en un período de tiempo (así que si están cambiando el comportamiento es difícil de predecir), los cálculos se llevaron a cabo durante un período de tiempo, y las salidas se escriben en un período de tiempo (por lo que si se leen demasiado pronto por la red o algún otro módulo que llama, el comportamiento es difícil de predecir).

  Visto desde el nodo, esto no tiene tiempo: Calcula basa en un único conjunto atómico de entradas (que piensa que se produjo a la vez), y reacciona instantáneamente.

  En el mismo momento un nodo síncrono se comunica con su entorno, que es el resto de la red, la comunicación es asíncrono. Incluso si ese nodo es el único en la red, si se comunica con sí mismo a través de la red asíncrona, esta será una comunicación asíncrona. Como resultado, la simulación de un conjunto de módulos de Esterel normalmente diferirá en el simulador Esterel (que supone la comunicación síncrona entre módulos) y el simulador de POLIS (que supone la comunicación asíncrona).Incluso con un único módulo, resultados de la simulación pueden ser diferentes.

- *Estados Infinitos*

  Una propiedad muy agradable de los sistemas sincrónicos Esterel y sistemas POLIS GALS es que tienen estados finitos, y por lo tanto el amplio conjunto de herramientas para el análisis y la optimización de los sistemas de estados finitos se pueden aplicar.

  En **Esterel,** el sistema de límite de estado a estado se basa en el impulso del reloj: el sistema cambia de estado en cada garrapata, y cualquiera de los dos programas sincrónicos son equivalentes si tienen el mismo comportamiento-tic-tic a. Las propiedades pueden ser verificadas con base en trayectorias de estado. Aunque dos sistemas pueden tener diferente micro-paso o micro-estado comportamiento, esto no es visible fuera de acuerdo con el modelo.

  En **POLIS**, el sistema de límite de estado a estado se basa en la ejecución basada en eventos de CFSMS: cada vez que se ejecuta una sola CFSM en hardware o software, se lee / computa / escribe y por lo tanto cambia el estado del sistema. Este es un nivel mucho más fino de granularidad, lo que resulta en sistemas con muchos más estados. Por otra parte, el comportamiento puede ser más difícil de predecir y controlar, ya que no hay reloj global que todos CFSMS son esclavo.

  Es evidente que hay una mayor flexibilidad en la aplicación con el modelo POLIS, mientras que el modelo Esterel proporciona un modelo que hace que sea más fácil de controlar y predecir el comportamiento del diseño. Por esta razón las implementaciones síncronas con frecuencia han sido elegidos para las aplicaciones críticas para la seguridad. Sin embargo, los requisitos de alta velocidad, y sistemas distribuidos heterogéneos deben utilizar un modelo asíncrono.

SIMULACION SINCRONICA Y GALS

Compararemos las simulaciones entre POLIS y Esterel de los mismos o similares sistemas para ilustrar las diferencias entre los modelos de comunicación totalmente sincrónica y GALS. De aquí en adelante, estamos considerando de manera efectiva una implementación particular de un programa escrito síncrona: una implementación de software completo con una semántica particular, para el medio ambiente que se describe a continuación.

Consideramos varios sistemas de módulos individuales (nodos) conectados a un módulo de entorno sencillo. En POLIS, dos C-archivos son generados por el sistema: una para el mismo módulo y una para el sistema operativo que es responsable de llamar al módulo cada vez que se dispara. La compilación de estos archivos se traduce en un simulador ejecutable. En Esterel, un solo C-archivo es generado por el módulo (lo mismo se aplica si hay un conjunto de módulos). Este archivo se compila y se enlaza a una biblioteca de simulación que se encarga de leer la entrada del usuario, llamando a todos los módulos de la ejecución, y la escritura de salida de usuario en cada instante

Para simular los sistemas, el usuario proporciona un conjunto de entradas y luego lee las salidas correspondientes. A esto le llamamos una *reacción de simulación*. En Esterel, esto es equivalente a una única instantánea. En POLIS, las reacciones son disparada por evento y no hay noción global de la instantánea (ya que el término instantánea se aplica sólo a un módulo síncrono sola, cada módulo tiene su propio horario de instantes). Una reacción de simulación es entonces la llamada sucesiva de todos los CFSMS que han sido disparadas, hasta que no haya CFSMS ejecutables restantes. En otras palabras, una reacción de simulación es un conjunto de instantes de CFSMS y, en particular, que puede incluir múltiples instantes o llamadas de un CFSM. (Esto es como un paso VHDL.) Es importante tener en cuenta este punto: la interfaz del simulador requiere que creamos un horario bastante artificial de puntos de tiempo en el que observamos el sistema.

En Esterel, todos los módulos reaccionan a cada instante, por lo que las nuevas salidas pueden ser producidas sin una entrada de disparo. En POLIS, todas las reacciones son activado por eventos, por lo que sin eventos de entrada, no se producirán eventos de salida (excepto en el caso en el que un módulo es auto-activación, lo que puede ser utilizado, por ejemplo, para asegurar que un evento ocurre después otra, para ordenar los eventos).

En Nuestro ejemplo, utilizamos el lenguaje Esterel como entrada para los dos Esterel y polis. Esterel tiene un conjunto bien definido de construcciones para escribir un programa que describe un sistema, incluyendo las declaraciones convenientes para la secuenciación, concurrencia, y la anticipación. La familia más importante de los estados en el lenguaje Esterel para nuestros propósitos son las relativas a retrasar. Una vez más, un retardo atómica en Esterel es un solo instante, mientras que un retraso atómica en POLIS es indefinida y depende de la implementación del sistema.

En otras palabras, esta declaración significa devolver el control al sistema operativo, pero asegúrese de que el control regresa a esta CFSM con el tiempo (esto se auto

desencadenando). Del mismo modo, la declaración espera SIG; en Esterel significa esperar hasta el siguiente instante que SIG está presente. En polis, que significa devolver el control al sistema operativo, y no desencadena este CFSM de nuevo hasta SIG está presente (que a su vez podría ser en la misma reacción de simulación).

Ejemplo

Si consideramos el siguiente modulo con su entorno de red circundante

```
module NODE:
input i, a_in;
output o, a_out;

await i;
emit a_out;
await a_in;
emit o
end module

net POLIS_NETWORK:
input i;
output o;
module NODE [i/i, a_in/a, a_out/a, o/o]
.

module ESTEREL_NETWORK;
input i;
output o;
signal a_in
run NODE [signal i/i, a/a_in, a/a_out, o/o]
end signal
end module
```

VI. CONCLUCION

Este trabajo académico ha sido realmente importante, y ha dado como fruto información acerca del marco para Hardware-Software Co-Diseño de Sistemas Embebidos que han servido de punto de partida para el presente artículo donde el sistema POLIS se compone de nodo del módulo y POLIS_NETWORK neta; el sistema de Esterel se compone de nodo del módulo y módulo de ESTEREL_NETWORK. Adjuntamos el comportamiento de interés por lo que puede desencadenar de forma explícita, por lo que podemos observar la forma de realización del cálculo. La red contiene la entrada y la salida i o, y una señal interna de una que se utiliza para conectar A_IN y A_OUT del NODO. Esta conexión, porque i es a través de la red, es asíncrona en POLIS y sincrónica en **Esterel** donde se especificó anterior mente.

En modelos y herramientas que polis maneja de co-diseño se diferencian por el nivel donde se puede dar la reprogramación. Cuando ésta se da en el nivel de aplicación de un procesador, el co-diseño consiste en el tratamiento unificado y la síntesis automática del software y del hardware, asumiendo este último como un co-procesador. Donde es de gran utilidad

Las diferencias en las dos simulaciones son entonces

1. El simulador para el modelo GALS ejecuta dos instantes CFSM en una reacción de simulación, ya que se ejecuta hasta que la quiescencia.

2. El sistema bajo el modelo de los galones emite de salida s; bajo el modelo puramente sincrónico o no se emite.

Por último, hay que tener en cuenta algunas diferencias simples entre Esterel y POLIS:

1. Esterel espera hasta el primer ciclo de reloj especificado por el usuario para generar su *primera* reacción. POLIS genera la reacción inicial en el arranque.

2. El formato interno en POLIS, llama cambio, tiene algunas limitaciones. Lo más importante para esta discusión, una señal que es tanto una entrada y de salida no pueden ser representados por el mismo nombre, y por lo tanto está representado por dos señales. Esto puede dar lugar a diferencias de comportamiento, aunque en la práctica las señales de Input Output no se deben utilizar en un módulo de POLIS, sino más bien conectados a nivel de red

VII. BIBLIOGRAFIA

- *https://embedded.eecs.berkeley.edu/Research/hsc/abstract.html#group*
- *http://embedded.eecs.berkeley.edu/research/hsc/self_guided_lab/lab_notes.html*
- *https://embedded.eecs.berkeley.edu/Research/hsc/polis_gals.html*
- *http://www.iberchip.net/VIII/docs/posters/p9.pdf*
- *http://bibing.us.es/proyectos/abreproy/11219/fichero/pfc_hgm.pdf*